The Adam optimisation algorithm

Thomas Salomon[1], Ismail Jamal-Eddine[2], and Ilies El Jaouhari[3]

[1] thomas.salomon@ensae.fr
[2] ismail.jamal-eddine@ensae.fr
[3] ilies.eljaouhari@ensae.fr

Keywords:

1 Introduction

The *Adaptive Moment Estimation* method was introduced by Diederik P. Kingma and Jimmy Ba [1]. It is an optimization algorithm that has gained prominence in the field of deep learning due to its ability to efficiently optimize neural network models. The need for Adam arises from the challenges posed by traditional optimization techniques like *Stochastic Gradient Descent* (SGD) and *RMSProp*, which often require careful manual tuning of hyperparameters and may struggle with issues like slow convergence and sensitivity to learning rates. Adam addresses these challenges by adapting the learning rates for each parameter during training, effectively combining the advantages of both momentum-based methods and adaptive learning rate methods. Its purpose it to help neural networks converge faster, handle noisy or sparse gradients, and mitigate issues related to saddle points or local minima.

Prior optimization techniques, notably *Stochastic Gradient Descent* (SGD) and *RMSProp*, have laid the foundation for neural network training. *SGD*, characterized by its simplicity and ease of implementation, is effective for a wide range of problems but often falls short in handling non-convex optimization tasks common in deep learning. Its main limitations include a tendency to get trapped in local minima and a slow convergence rate, particularly in the presence of noisy or sparse gradients. *RMSProp*, an extension of *SGD*, improves upon this by adapting the learning rates based on the recent magnitude of gradients, mitigating the issue of vanishing or exploding gradients. However, *RMSProp* still struggles with the need for manual fine-tuning of hyperparameters and is less effective in scenarios involving very noisy or sparse data. In contrast, the *Adam* algorithm advances these methods by intelligently combining the benefits of momentum (from *SGD* with momentum) and adaptive learning rate (from *RMSProp*), leading to more robust and efficient convergence in practice. This adaptability allows it to perform consistently across various types of neural networks and datasets, making it a versatile tool in the machine learning practitioner's toolkit. Additionally, *Adam*'s ability to handle saddle points—where gradients are flat, which is a challenge for traditional methods—further underscores its effectiveness in complex optimization landscapes typical in deep learning applications.

In this project, we chose to dive into the Adam optimization algorithm to understand its intricacies and assess its performance, aiming to uncover its strengths and weaknesses in comparison to more traditional optimization techniques.

2 Description

Since the inception of Neural networks, researchers have looked for reliable and robust optimization techniques because the ones available had troubles adapting to the large varieties of architectures they could offer. One of the first to stood out was the RMSProp with its use of exponential moving averages to correct the gradient's second moment estimation. Adam improves the latter by making use of a momentum component to help the process gain in robustness.

The algorithm differs from its famous predecessor RMSProp[2] by using exponential moving averages of the first of the stochastic gradient. This update allows us to build a certain momentum when looking the global minima of the function we are trying to optimize. The exponential moving average adds a fraction β_1 of the previous updates to the new one. If we moved a lot toward a certain direction, the new updates will incorporate part of the previously high velocity. The analogy also works the way around: if we make little steps before the update at date t, seeing a high gradient (relative to the previous values) will likely takeover the previous steps in terms of magnitude and "lead" the historical average. Thus, the vector $\beta = (\beta_1, \beta_2)$ controls how much memory we retain at each new update, up until this innovation most methods used to retain only the current gradient. The following pseudo-code describes the algorithm:

Algorithm 1 Adam: Stochastic Optimization

```
Require: Step size $\alpha$
Require: Exponential decay rates for the moment estimates $\beta_1, \beta_2 \in [0, 1)$
Require: Stochastic objective function with parameters $f(\theta)$
Require: Initial parameter vector $\theta_0$
Initialize 1st moment vector $m_0 \leftarrow 0$
Initialize 2nd moment vector $v_0 \leftarrow 0$
Initialize timestep $t \leftarrow 0$
while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    Get gradients w.r.t. stochastic objective at timestep $t$: $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    Update biased first moment estimate: $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
    Update biased second raw moment estimate: $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
    Compute bias-corrected first moment estimate: $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
    Compute bias-corrected second raw moment estimate: $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$
    Update parameters: $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$
end while
return Resulting parameters $\theta_t$
```

At each update before the convergence of θ_t, the momentum step corresponding to the update of the first moment is applied, followed by that of the second moment which comes from the innovation of RMSProp[3]. The last steps of the process constitute the bias correction which helps with the potential instability or divergence in the early stages of training because of the initialization of the moments' estimates at 0 at the beginning of the algorithm.

3 Preliminary Analysis

The empirical analysis of the efficiency of the Adam optimization algorithm started with a preliminary analysis employing synthetic data to gauge its accuracy. In this matter, we generated two distinct types of synthetic datasets: the first, a multi-modal dataset, was produced using the *make-gaussian-quantiles* function. This function is adept at generating the sort of data encountered in real-world situations, characterized by distinct clusters with varying central tendencies. It accomplishes this by distributing data points across a series of Gaussian distributions, with each distribution denoting a separate class.

The complex synthetic dataset was created using the *make-classification* function, which infuses the data with a greater level of complexity. This dataset is marked by the addition of many features, with only some providing informative insights into the class labels, while the rest serve as redundant or noise-inducing elements. Such attributes render this dataset a more suitable challenge for the Adam optimization algorithm due to its high dimensionality and the intentional introduction of noise, mirroring the often imperfect and noisy conditions inherent in real-world datasets.

In the second phase of our study, we applied the Adam optimization algorithm within a neural network model to both the multi-modal and complex synthetic datasets. The goal during this phase was to assess how the neural network, powered by Adam's optimization capabilities, managed to learn and perform on these datasets. Throughout the training, the model's parameters were adjusted in accordance with Adam's optimization strategy to minimize classification errors. This iterative updating process spanned multiple epochs, with each epoch constituting a complete cycle through the full dataset. The model's proficiency in accurately identifying true clusters within the synthetic datasets was subsequently evaluated using a confusion matrix. This instrument offered a clear visual representation of the model's classification accuracy, providing a detailed breakdown of its performance by highlighting the true positive and false positive rates for each class. Through this comprehensive approach, we sought to affirm the potential of the Adam optimization algorithm as a robust tool for neural network training, capable of handling the intricate patterns and noise characteristic of complex data structures.

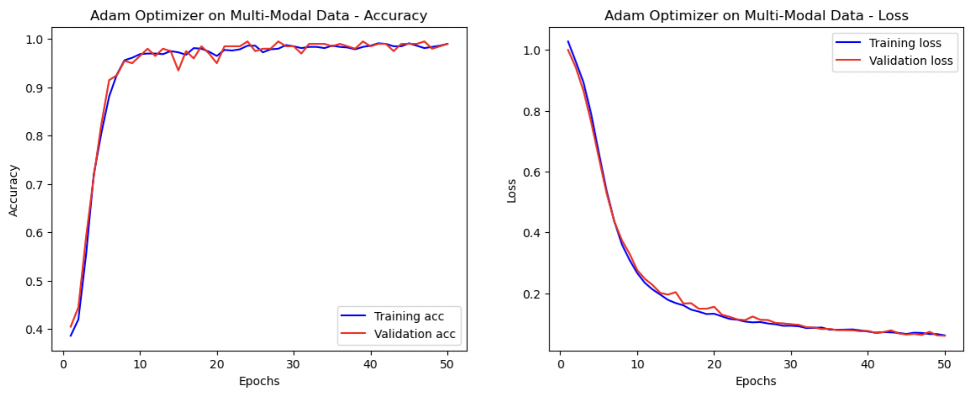


Figure 1: Adam Optimizer Performance Metrics on Multi-Modal Dataset

3.1 Performance on Multi-modal Synthetic Dataset

The Adam optimizer's application to the multi-modal data-trained model demonstrates commendable efficacy, as reflected by the high accuracy rates and diminished loss levels. Consistency in the accuracy and loss progressions throughout the training indicates a robust learning process. This robustness also suggests that the selected learning rate and other hyperparameters for the Adam optimizer are well-suited for this specific task. The inherent structure of the multi-modal dataset, characterized by its distinct and separate clusters, naturally lends itself to more straightforward neural network learning processes, thereby explaining the observed high performance.

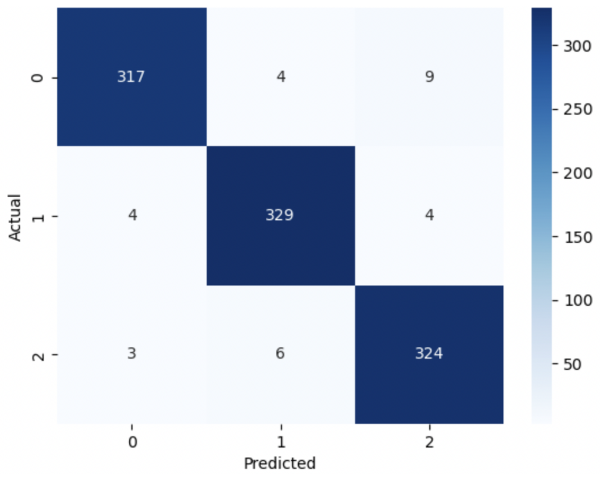


Figure 2: Classification Performance of Adam Optimizer on Complex Synthetic Dataset

3.2 Performance on Complex Synthetic Dataset

Upon reviewing the confusion matrix for the model leveraging the Adam optimizer on the complex dataset, we observe a robust classification performance. The matrix details that Class 0 sees a majority of its instances accurately classified, with a minor count misclassified as Class 1 or Class 2. Class 1 demonstrates commendable classification precision, with a small number of instances confused as Class 0 or Class 2. Class 2 similarly shows strong classification accuracy, though there are a few instances where Class 1 is mistaken for Class 2 and vice versa. These outcomes highlight the Adam optimizer's proficient handling of the dataset's intricate feature space, affirming its capability to differentiate between clusters despite the data's inherent complexity and injected noise. The confusion matrix, thus, reflects a model that is well-tuned and capable of effective classification, even under the challenge of a high-dimensional and noisy dataset.

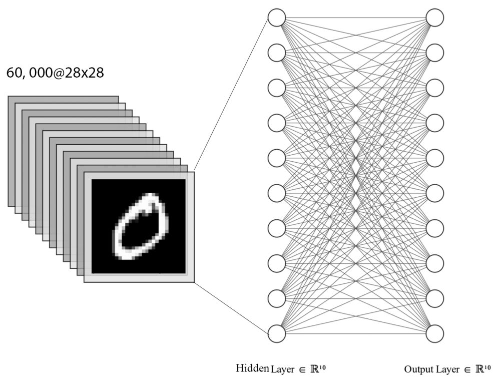
The initial evaluation of our optimizer highlights its competence in accurately identifying the true clusters within synthetic datasets. Nonetheless, to fully evaluate its effectiveness, a further assessment using the MNIST dataset is essential, as this will provide insight into its performance on a dataset with a higher level of complexity.

4 Implementation

4.1 Architecture

report (except for the comparative analysis) use a standard two-layer feed forward architecture. The layer configuration is composed of an input layer combined with a ten-neuron hidden layer and output layer. The first uses a ReLU activation function for the mitigation of the vanishing gradient as verifed in [4] and also for its well-known computational effiency. For the output layer, a softmax function was used which is a fairly standard choice for our type of problem. The neural network architecture schematics is presented below for clarity:

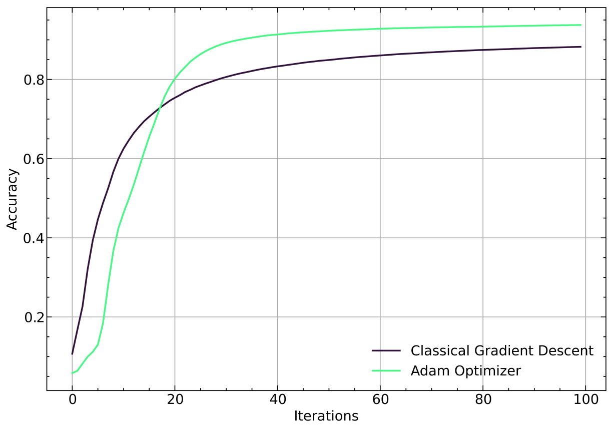
Figure 3: Two-layer feedforward architecture



4.2 First results

On this figure, we analyze the accuracy trends of the Adam optimizer compared to a classical approach, distinct patterns emerge that are noteworthy for our project report. Initially, Adam starts with a lower accuracy of approximately 5.83% but exhibits a rapid improvement, reaching about 42.4% by the 100th iteration. In contrast, the classical optimizer begins with a slightly higher accuracy of around 10.72%, yet it demonstrates a more gradual increase, achieving close to 39.9% in the same number of iterations.

Figure 4: Accuracy comparison



As the training progresses further, particularly noticeable up to the 500th iteration, Adam's efficiency becomes more evident. The accuracy under this optimizer sharply increases to approximately 92.2%, significantly outpacing the classical optimizer, which attains around 88.1% accuracy. This phase of training highlights Adam's capability for faster convergence, benefiting from adaptive learning rates and momentum. In the latter stages of training, extending to the 1000th iteration, both optimizers show signs of converging, with incremental improvements in accuracy. Adam maintains its lead, culminating at an accuracy of about 93.8%, while the classical optimizer concludes with a final accuracy of roughly 88.3%.

From this analysis, it is evident that the Adam optimizer is particularly effective in rapidly navigating towards higher accuracies, especially in the initial and mid-phases of the training process. Its structure, which intelligently adjusts learning rates and incorporates momentum, contributes to this enhanced performance. In contrast, the classical optimizer, while being simpler and more straightforward, does not quite match the acceleration and peak performance of Adam.

5 Comparative analysis

Adam algorithm is the combination of both RMSProp and Adagrad, in this section we will compare those algorithms and Adadelta to Adam with respect to the loss and the accuracy of our model. RMSProp stands for Root mean square propagation, it was one of the first algorithm that implemented an adaptative learning rate for neural networks. It was based on R Resilient Back Propagation, this algorithm is the bed rock of neural networkS. It enabled, thanks to chain rule to update each weight in the network with respect to an arbitrary error.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial s_i} \cdot \frac{\partial s_i}{\partial net_i} \cdot \frac{\partial net_i}{\partial w_{ij}} \tag{1}$$

$$w_{ij}(t+1) = w_{ij}(t) - \varepsilon \cdot \frac{\partial E}{\partial w_{ij}}(t) \tag{2}$$

$$\Delta w_{ij}(t) = -\varepsilon \cdot \frac{\partial E}{\partial w_{ij}}(t) + \mu \Delta w_{ij}(t-1) \tag{3}$$

- w_{ij}: weight from neuron j to neuron i
- s_i: output of neuron i
- net_i: weighted sum of the inputs of neurons i
- ε: learning rate
- μ: momentum parameter

However the steps to update the weights are to small and therefore it was time consuming to train an neural network with a large dataset. RMSProp was designed to tackle the problem of vanishing learning rate. This algorithm use the square of the gradient:

$$E[g^2](t) = \beta E[g^2](t-1) + (1-\beta)\left(\frac{\partial c}{\partial w}\right)^2 \tag{4}$$

$$w_{ij}(t) = w_{ij}(t-1) - \frac{\eta}{\sqrt{E[g^2](t)}} \frac{\partial c}{\partial w_{ij}} \tag{5}$$

AdaGrad is another method to optimise the gradient descent. AdaGrad is the short for Adaptative learning rates, it was first introduced in 2011. The idea is to have an adaptative learning rates in order to hasten convergence. The main objective was to tackle the problem of sparse gradient. This is the main objective as RMSProp, find the good balance for adapted steps. The learning rate update has been designed in such way to it will automatically decrease, because of the summation of the previous gradient as you can see in the pseudo code bellow:

Algorithm 2 Adagrad: Adaptive Gradient Algorithm

```
Require: Global learning rate $\eta$
Require: Initial parameter vector $\theta$
Require: $\epsilon$ (often set to $1e-8$) to prevent division by zero
Require: Stochastic objective function with parameters $f(\theta)$
  Initialize gradient accumulation vector $G_0$ as zero vector
  Initialize timestep $t \leftarrow 0$
  while $\theta_t$ not converged do
      $t \leftarrow t+1$
      Get gradient stochastic objective at timestep $t$: $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
      Accumulate squared gradients: $G_t \leftarrow G_{t1}+g_t \odot g_t$ ($\odot$ denotes element-wise multiplication)
      Compute update: $\Delta\theta_t \leftarrow -(\eta/(\sqrt{G_t}+\epsilon)) \odot g_t$
      Apply update: $\theta_t \leftarrow \theta_{t-1}+\Delta\theta_t$
  end while
  return $\theta_t$
```

Adadelta is another algorithm derived from Adagrad. Whereas using bluntly the sum of square gradient, it uses an average of the square gradient :

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2 \tag{6}$$

$$\Delta\theta_t = -\eta \cdot g_{t,i} \tag{7}$$

$$\theta_t = \theta_{t-1} + \Delta\theta_t \tag{8}$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t \tag{9}$$

In this graph below we can clearly see, that the Adam optimizer is the best both in term of

accuracy and loss function. For the sake of clarity we did not display RMSProp in the plot, it was behaving a bit erraticly.It might be caused by the fact that RMSProp was designed for ANN (Artificial Neural Network). For Adadelta γ behave just like the β for Adam, so we usually set $\gamma = 0.95$. We can the that see that Adagrad converge more quickly than Adadelta, it might be caused by the agressivity of Adagrad. It performs well to predict "easy to go" minima, but can be trap in local minima, whereas the exponential mean in Adadelta smooth the convergence and perform better real life data with sparse gradient.

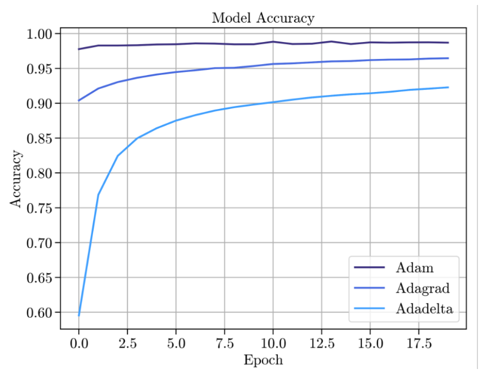


Figure 5: Accuracy vs. Iterations for Different Beta Values

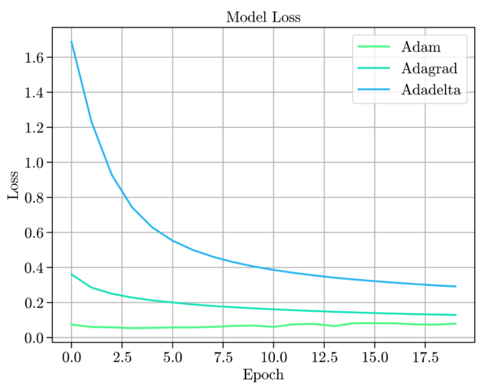


Figure 6: Loss vs. Iterations for Different Beta Values

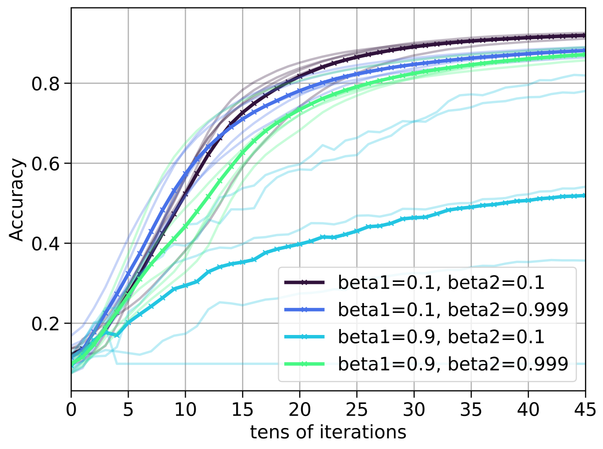
6 Hyper-parameter exploration

The authors of [1] offer general recommendations in regard with the optimisation parameters. We recall here each parameters in the algorithm and how they affect it:

- α is the learning rate of the process. The authors do recommend tuning this parameter for better performance. However in our case, we do not find the tuning useful because of the limited interpretations we could draw from the results. In the following example, the value that we have kept is $\alpha = 10^{-3}$
- β_1 controls the momentum component and the exponential decay of the first moment estimate.
- β_2 is inherited from RMSProp and is particularly useful in helping the optimization process deal with non-stationary objectives.
- ϵ allows to manage the specific case where we have very low values for the second moment estimate causing explosion of the adaptative learning rate. This parameter does not really affect performances and the authors of [1] make the recommendation of keeping it to 10^{-8}.

The figure below illustrates how the accuracy evolves during the optimization process for different parameter combinations (β_1, β_2), utilizing a 5-fold cross-validation approach. Each line represents the accuracy trajectory for a particular fold, depicted as transparent lines, while the more opaque lines denote the average accuracy across all folds for each (β_1, β_2) set. This method of cross-validation provides a more robust understanding of the model's performance under different hyperparameters by averaging across multiple training-validation splits:

Figure 7: Accuracy vs. Iterations for Different Beta Values



This plot makes it obvious that there is a skewed interaction between the components of β meaning that the update of the first and the second moments of our stochastic gradients do not operate in an independent manner. We can see that the combination $\beta_1 = 0.9, \beta_2 = 0.1$ clearly shows a rougher profile compared to the others highlighting the fact that the SGD navitages in a non-smooth way on the rugged loss surface. This combination corresponds to a asymmetric memory setup where we retain the majority of information from the past updates for the gradient's first moment but only 10% (at each update) for the second one underlining the necessity of the RMSProp step mentioned in 2. Adding this observation to one about the curve representing $\beta = (0.1, 0.1)$ highlights the importance of the term $\frac{\alpha}{\sqrt{\hat{v}_t}+\epsilon}$ and how we adapt the learning rate depending on the past measurements of the gradient's variance with a high momentum component show the important role of the RMSProp in Adam.

For the remaining values of β, we can retrieve insights about the loss landscape from the fact that $\beta = (0.1, 0.1)$ and $\beta = (0.1, 0.999)$ seem to outperform the values advised by the authors in [1]. Since we do not prioritize the past gradient updates in the exponential reweighing we can hypothesize that the more complex, non-convex, or rapidly changing loss landscape where the assumptions of smoothness and gradual change (which higher β_1 and β_2 values rely on) are not entirely valid. It also suggests that for our specific problem involving the MNIST dataset, the strengths of Adam in terms of balancing momentum and adaptability are best leveraged with a configuration that emphasizes immediate gradient information over historical averages.

7 Conclusion

This report, initially aimed at presenting the strengths and weaknesses of the Adaptive Moment Estimation algorithm, presented its innovations compared to its famous predecessor RMSProp. The comparison with the popular Adagrad, both in terms of theorical structure and practical results, revealed significant and superior loss for Adam. We could see by the benchmark on different multi-modal datasets (both synthetic and real-world) how it outperforms its predecessors by implementing additional robust components: momentum and bias correction for first and second moments. The effect of those was explored by tuning the hyper-parameters controlling them. We showed that non-default values of the vector β, which controls the memory by incorporating past updates of moments in an exponential averaging way, could handle different loss landscapes. The analysis highlighted in clear manner how the process defines a balance between momentum and adaptability.

References

[1] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[2] T. Tieleman and G. Hinton. Rmsprop, 2012.

[3] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[4] Leni Ven and Johannes Lederer. Regularization and reparameterization avoid vanishing gradients in sigmoid-type networks. *CoRR*, abs/2106.02260, 2021.